Physical model based systems design

Albert Benveniste (Inria-Rennes)

Updated October 2016
Contents

1. Physical Modeling & Systems Design: a vision

2. The foundations for compiling Modelica (multi-mode DAE systems)

Acknowledgements:
• Benoît Caillaud, Khalil Ghorbal
• Tim Bourke, Marc Pouzet;
• Hilding Elmqvist, Martin Otter, Sven-Erik Mattsson;
• Paul Caspi and Paul Le Guernic.
Physical Modeling & Systems Design: the overall vision
Modelica: multi-mode DAE systems:

\[
\begin{cases}
  \text{if } b \text{ do } F(\dot{x}, x, u) = 0 \\
  b = \text{BoolCond}(\dot{x}, x, u)
\end{cases}
\]

- Modelica supports component based physical system modeling (not Simulink)

- Compilation is complex:
  - Latent constraints
  - Index reduction
  - Structural analysis

- Requires sophisticated causality analyses (as for bond graphs)
Not discussed see [Elmqvist 2014, 2015]

3D CAD

Our focus

Requir.
Safety Monitoring
Modelica & Requirement Engineering
A requirement profile has been defined for Modelica [Fritzson14]

- Provision for writing requirements
- Linking requirements to test cases
- The link is syntactic, not semantic

Is this all we need? No!

Requirement architectures differ from (physical) system architectures
Is this all we need? No!

Requirement architectures differ from (physical) system architectures

- Responsibilities must be clearly identified:
  - guarantees, vs.
  - assumptions

- Conjunction of requirements; several viewpoints

- From system to subsystem: refinement & parallel composition

\[ \text{CONTRACTS} \]
Modelica + Contracts

• Assume/Guarantee contracts

• \( C = (A, G) = (\text{Assumption, Guarantee}) = \text{pair of Modelica properties} \)

• All the needed operators and relations exist

• Other forms of contract exist...
Modelica & Safety Engineering
• Extend Modelica models with failure modes
• Use Modelica structural analysis to derive fault effects and propagation (fault tree)
• Check critical branches of the fault tree on the detailed Modelica model (guided simulation)

We can go beyond and perform system wide alarm handling
Network of automata modeling fault propagation
Diagnosis algorithm

- Automaton describing the operating modes (nominal, failed_x, ...) and their transitions
- Some transitions are observed (alarms)
- System = product of many such automata
- Reconstruct hidden state histories from observations (state observer)
How to construct models?

Self-Modeling

Capturing architecture (network discovery)

Standards:
SDH, WDM, OTN, GMPLS...

Behavior of generic Network elements

Automatic generation and deployment of diagnosis algorithms

Automatic generation of behavioral model
How to construct models?

Self-Modeling

Capturing architecture (network discovery)

Behavior of generic elements

Automatic generation of behavioral model

Automatic generation and deployment of diagnosis algorithms

FMEA (Fault Mode and Effects Analysis)

traceability + models
Modelica & System wide Diagnosis
Modelica supports component based physical system modeling (not Simulink)

Compilation is complex:
- Latent constraints
- Index reduction
- Structural analysis...

Requires sophisticated causality analyses (as for bond graphs)

Idea: exploit the power of Modelica analyses by automatically deriving parity checks
From Modelica to parity checks, automatically

- Westinghouse braking system; control: pressure at the head of the train
- Each wagon induces two modes: valve $D_1$ open / closed
  - $2^n$ modes for a $n$ wagons train
- Resistor $R_3$ captures possible leakage
Westinghouse braking system; control: pressure at the head of the train

Each wagon induces two modes: valve $D_1$ open / closed
- $2^n$ modes for a $n$ wagons train

Resistor $R_3$ captures possible leakage
- Nominal / Leak: $R_3 = \infty / R_3 < \infty$

Goal: monitoring for a possible leakage
- What should we measure?
- Where to put sensors?
- Getting all of this from Modelica compilation
• The failure is non detectable when $D_1$ is open (no breaking mode)
  • (no flow traverses $R_3$ in this case)
  • Diagnosticability is mode-dependent (recall: $2^n$ modes for a $n$ wagons train)

• How to generate parity checks
  • To monitor all possible leaks
  • By measuring (some or all of) the flows?
Idea: reuse the same Modelica model with the following adjustments:

- Subset of the flows $\varphi_{j_1}, ..., \varphi_{j_k}$: inputs (possibly constrained)
- Resistors $R_1, ..., R_n$: nominal parameters
- Unobserved states $X = (x_1, ..., x_m)$

The mode-dependent causality analysis of the Modelica model reveals that diagnosticability is mode-dependent.
From Modelica to parity checks, automatically

- We have our Modelica model for simulation
- And the actual system for monitoring
- Some (but not all) states or outputs are measured
From Modelica to parity checks, automatically

- And feed the Modelica model with all the measurement data
- Yields an overconstrained Modelica model; exploit it to measure model/data fit
- Collect measurement data from the system in operation
Contents

1. Physical Modeling & Systems Design: a vision

2. The foundations for compiling Modelica (multi-mode DAE systems)
The need for flexibility and solid foundations
Modelica, thou shall be flexible and formally sound

• **Flexible**
  - Simulating
  - Supporting safety analyses
  - Generating fault trees
  - Generating parity equations
  - Handling multi-mode with no restriction
  - Supporting non-regular systems?

• **Formally sound**
  - Benefiting from the heritage of synchronous languages
Challenging hybrid causal loops in Modelica tools

model scheduling
  Real x(start=0);
  Real y(start=0);

equation
  der(x)=1;
  der(y)=x;

  when x>=2 then
    reinit(x,-3*pre(y));
  end when;
  when x>=2 then
    reinit(y,-4*pre(x));
  end when;
end scheduling

At the instant of reset, x and y each have a value defined in terms of their values just prior to the reset.
Challenging hybrid causal loops in Modelica tools

model scheduling
  Real x(start=0);
  Real y(start=0);

  equation
    der(x)=1;
    der(y)=x;

    when x>=2 then
      reinit(x,-3*y);
    end when;
    when x>=2 then
      reinit(y,-4*x);
    end when;

end scheduling

Take the pre’s away:

At the time of reset, $x$ and $y$ are in cyclic dependency chain.

The simulation runtime (of both OpenModelica and Dymola), chooses to reinitialize $x$ first, with the value $-6$ as before, and then to reinitialize $y$ in terms of the updated value of $x$: $24$. 
Challenging hybrid causal loops in Modelica tools

What happens, if we reverse the order of the two reinit?...

The simulation result changes, as shown on the bottom diagram.

The same phenomenon occurs if the reinit’s are each placed in their own when clause.

```
model scheduling
  Real x(start=0);
  Real y(start=0);

equation
  der(x)=1;
  der(y)=x;

  when x>=2 then
    reinit(x,-3*y);
  end when;

  when x>=2 then
    reinit(y,-4*x);
  end when;

end scheduling
```
The causal version (with the “pre”) is scheduled properly and simulates as expected.

The non-causal programs are accepted as well, but the result is not satisfactory.

Algebraic loops cannot be rejected, even in resets, since they are just another kind of equation. They should be accepted, but the semantics of a model must not depend on its layout!

Studying causality can help to understand the detail of interactions between discrete and continuous code.
All about synchronous languages in a few slides

Compilation schemes from the Constructive Semantics
An example of Signal program and its compilation

• **Why discussing Signal?**
  • Among synchronous languages, Signal is closest to Modelica
  • It has clocks and equations on clocks, and
  • Requires mode-dependent causality analysis
An example of Signal program and its compilation

- **Why discussing Signal?**
  - Among synchronous languages, Signal is closest to Modelica
  - It has clocks and equations on clocks, and
  - Requires mode-dependent causality analysis

- **The Signal vintage watch**
  - This is an old mechanical watch like the one I have. Turn the button. The watch goes for some time, and then stops. When it stops, turn again the button... and so on...
An example of Signal program and its compilation

( X := IN default ZX-1
| ZX := X$1 init 0
| IN ^= when (ZX < 0) )

Input IN returns X

This was Signal code; Lustre-like pseudo-code follows:

pre(X) init 0 in
if pre(X) < 0
    then (get IN and set X := IN)
else (set X := pre(X)-1)
An example of Signal program and its compilation

\[ ( X := \text{IN default ZX-1} \]
\[ | \ ZX := X \$$1 \text{ init 0} \]
\[ | \ B := (ZX < 0) \]
\[ | \ \text{IN} ^\wedge = (\text{when B} ) ^< B \]
\[ | \ H ^\wedge = B ^\wedge = X ^\wedge = ZX ) \]

\[ [B] : \text{when B} \]
An example of Signal program and its compilation

( X := IN default ZX-1
| ZX := X$1 init 0
| B := (ZX < 0)
| IN ^= (when B) ^< B
| H ^= B ^= X ^= ZX )

[B]: when B

Case B = true
Case B = false
An example of Signal program and its compilation

( X := IN default ZX-1
  | ZX := X$1 init 0
  | B := (ZX < 0)
  | IN ^= (when B) ^< B
  | H ^= B ^= X ^= ZX )

[B]: when B

Case B = true
An example of Signal program and its compilation

```
(  X := IN default ZX-1
| ZX := X$1 init 0
| B := (ZX < 0)
| IN ^= when B ^< B
| H ^= B ^= X ^= ZX )
```

[B]: when B

Case B = false
An example of Signal program and its compilation

**Constructive Semantics:**

- execution scheme that schedules
- atomic actions *(here: evaluating expressions)*
- and successfully evaluates all variables at each reaction
From Synchronous Languages to the Structural Analysis of multi-mode DAE systems

From continuous to discrete time using non-standard analysis
A simple clutch

- The clutch has two modes:
  - engaged: $\gamma = T$; DAE
  - released: $\gamma = F$; ODE

\[
\begin{align*}
\omega_1' &= f_1(\omega_1, \tau_1) \\
\omega_2' &= f_2(\omega_2, \tau_2)
\end{align*}
\]

\[
\begin{align*}
\text{if } \gamma &\text{ then } \\
\omega_1 - \omega_2 &= 0 \\
\tau_1 + \tau_2 &= 0
\end{align*}
\]

\[
\begin{align*}
\text{else } \\
\tau_1 &= 0 \\
\tau_2 &= 0
\end{align*}
\]
A simple clutch

- The clutch has two modes:
  - engaged: $\gamma = T$; DAE
  - released: $\gamma = F$; ODE

- Is it enough to make DAE analysis mode dependent?
  - problem: this says nothing about how to handle the resets at mode change

- When the clutch engages and the two rotation speeds differ, an impulse occurs for the torques

- This example is not supported by existing Modelica tools today
A simple clutch
the “engaged” mode

\[
\begin{align*}
\omega'_1 &= f_1(\omega_1, \tau_1) \\
\omega'_2 &= f_2(\omega_2, \tau_2)
\end{align*}
\]

if $\gamma$ then
\[
\begin{cases}
\omega_1 - \omega_2 = 0 \\
\tau_1 + \tau_2 = 0
\end{cases}
\]

else
\[
\begin{cases}
\tau_1 = 0 \\
\tau_2 = 0
\end{cases}
\]

- The source DAE model is in black
- In red I have added a latent equation, which implicitly holds although not written in the source
- When all latent equations are added (here only 1), we inherit a structurally nonsingular system of algebraic eqns $(e_1, e_2, e_4, e_5)$ with dependent variables $(\tau_1, \tau_2, \omega'_1, \omega'_2)$ (dummy derivatives)
- Solving it yields the velocities as an implicit function of the positions $\approx$ODE
A simple clutch
trying existing tools

\[
\begin{cases}
\omega_1' = f_1(\omega_1, \tau_1) \\
\omega_2' = f_2(\omega_2, \tau_2)
\end{cases}
\]

if \( \gamma \) then
\[
\begin{cases}
\omega_1 - \omega_2 = 0 \\
\tau_1 + \tau_2 = 0
\end{cases}
\]

else
\[
\begin{cases}
\tau_1 = 0 \\
\tau_2 = 0
\end{cases}
\]

\[
\begin{cases}
\omega_1' = f_1(\omega_1, \tau_1) \quad (e_1) \\
\omega_2' = f_2(\omega_2, \tau_2) \quad (e_2) \\
\omega_1 - \omega_2 = 0 \quad (e_3) \\
\omega_1' - \omega_2' = 0 \quad (e_4) \\
\tau_1 + \tau_2 = 0 \quad (e_5)
\end{cases}
\]

- Unfortunately, this tells nothing about how to handle the mode changes
- The difficult case is \( \gamma: F \to T \) (the clutch gets engaged)
- Some simulation results for this example by existing tools follow
A simple clutch trying Modelica

• Mode changes $F \rightarrow T \rightarrow F$ at $t = 5, 10$

The following error was detected at time: 5.002
Error: Singular inconsistent scalar system for $f_1 = ((\text{if } g \text{ then } w_1-w_2 \text{ else } 0.0))/(-(\text{if } g \text{ then } 0.0 \text{ else } 1.0)) = -0.502621/-0$
Integration terminated before reaching "StopTime" at $T = 5$

• The reason is that Dymola has symbolically pivoted the system of equations, independently of the mode. By doing so, it has produced an equation defining $f_1$ that is singular in mode $g$. 

model ClutchBasic
parameter Real w01=1;
parameter Real w02=1.5;
parameter Real j1=1;
parameter Real j2=2;
parameter Real k1=0.01;
parameter Real k2=0.0125;
parameter Real t1=5;
parameter Real t2=7;
Real t(start=0, fixed=true);
Boolean g(start=false);
Real w1(start = w01, fixed=true);
Real w2(start = w02, fixed=true);
Real f1;
Real f2;
equation
  der(t) = 1;
g = (t >= t1) and (t <= t2);
j1*der(w1) = -k1*w1 + f1;
j2*der(w2) = -k2*w2 + f2;
0 = if g then w1-w2 else f1;
f1 + f2 = 0;
end ClutchBasic;
A simple clutch trying Mathematica (NDSolve)

\[
\text{NDSolve[}
\begin{align*}
&\text{w1}'[t] == -0.01 \, \text{w1}[t] + \text{t1}[t], \\
&2 \, \text{w2}'[t] == -0.0125 \, \text{w2}[t] + \text{t2}[t], \\
&\text{t1}[t] + \text{t2}[t] == 0, \\
&\text{s}[t] \, (\text{w1}[t] - \text{w2}[t]) + (1 - \text{s}[t]) \, \text{t1}[t] == 0, \\
&\text{w1}[0] == 1.0, \text{w2}[0] == 1.501, \text{s}[0] == 0, \\
&\text{WhenEvent}[t == 5, \\
&\{\text{s}[t] \to 1\}]
\end{align*}
\], \\
\{\text{w1, w2, t1, t2, s}\}, \\
\{t, 0, 7\}, \text{DiscreteVariables} \to \text{s}
\]

No crash at mode change. But nondeterministic reset reveals that cold restart is indeed performed by NDSolve on this example.

Mode changes
\[F \rightarrow T \text{ at } t = 5\]
\[T \rightarrow F \text{ at } t = 10\]
A simple clutch
a comprehensive approach

\[
\begin{align*}
\omega'_1 &= f_1(\omega_1, \tau_1) \\
\omega'_2 &= f_2(\omega_2, \tau_2)
\end{align*}
\]

if \( \gamma \) then
\[
\begin{cases}
\omega_1 - \omega_2 = 0 \\
\tau_1 + \tau_2 = 0
\end{cases}
\]
else
\[
\begin{cases}
\tau_1 = 0 \\
\tau_2 = 0
\end{cases}
\]

• The difficult case is \( \gamma: F \rightarrow T \) (the clutch gets engaged)

• We handle this by invoking nonstandard analysis and expand:

\[
\omega' = \frac{\omega - \omega^\ddagger}{\partial}
\]

where \( \omega^\ddagger \) is the “next” operator and time step \( \partial \) is an infinitesimal of nonstandard analysis.

• This brings the whole model to discrete-time and we are able to combine the techniques from synchronous languages with those of index analysis from DAE.
The reset is handled satisfactorily. The rotation speed right after engagement sits between the two rotation speeds before, which matches the intuition from physics.

\[
\begin{align*}
\omega'_1 &= f_1(\omega_1, \tau_1) \\
\omega'_2 &= f_2(\omega_2, \tau_2)
\end{align*}
\]

if \( \gamma \) then
\[
\begin{align*}
\omega_1 - \omega_2 &= 0 \\
\tau_1 + \tau_2 &= 0
\end{align*}
\]

else
\[
\begin{align*}
\tau_1 &= 0 \\
\tau_2 &= 0
\end{align*}
\]

Mode changes
\( F \to T \) at \( t = 5 \)
\( T \to F \) at \( t = 10 \)
Structural Analysis of multi-mode DAE systems

See my detailed lecture
1. Physical modeling is central to systems design
   • Modeling for simulation
   • Modeling fault propagation
   • Generating parity checks for diagnostics
   • Complemented with modeling the computing architecture
1. Physical modeling is central to systems design
   • Modeling for simulation
   • Modeling fault propagation
   • Generating parity checks for diagnostics
   • Complemented with modeling the computing architecture

2. The compilation of physical models requires a difficult structural analysis
   • Source of difficulties in current tools
   • Techniques from synchronous languages help
   • Efficient algorithms are yet to obtain
Thanks